EA075
Processadores Dedicados Padronizados: Periféricos

Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (UNICAMP)

Prof. Rafael Ferrari

(Documento baseado nas notas de aula do Prof. Levy Boccato)

Introdução

- **Processador dedicado:** sistema digital projetado para realizar uma tarefa específica de computação.

- Algumas tarefas de computação são tão comuns que **processadores dedicados padronizados** (também chamados de periféricos) estão disponíveis como "itens de prateleira".

Motivação

- Um projetista de sistemas embarcados pode escolher utilizar um processador dedicado padronizado para implementar parte de uma funcionalidade desejada do sistema e, com isso, obter alguns benefícios:
  - O custo NRE é relativamente baixo - o processador já está pronto, faltando sua integração.
  - O custo por unidade do processador pode ser baixo – o NRE foi amortizado pela grande quantidade de unidades produzidas pelo fabricante.
  - Em relação a um processador genérico, o desempenho pode ser mais rápido, com consumo de potência menor e ocupando um tamanho reduzido – afinal, o processador dedicado foi criado para esta única tarefa.
- **Compromisso:** se um processador genérico já fizer parte do sistema embarcado em desenvolvimento, implementar uma tarefa através de um processador dedicado adicional em vez de software pode aumentar o tamanho e o consumo de potência do sistema.

Temporizadores / Contadores

- **Temporizador:** circuito capaz de medir intervalos de tempo.

  - Pode gerar eventos temporais – por exemplo, para manter o sinal verde em um semáforo por 10 s.

  - Pode medir o tempo decorrido entre a ocorrência de eventos – por exemplo, para computar a velocidade de um carro.

Temporizadores / Contadores

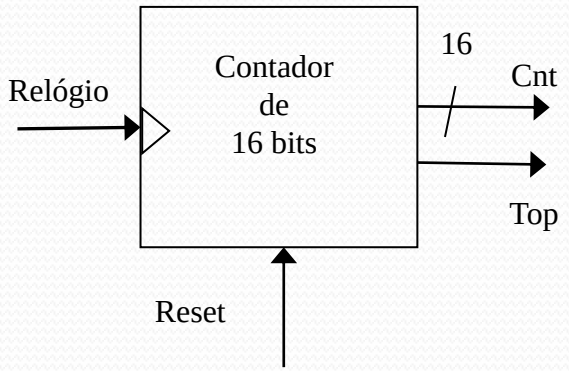
- **Como medir o tempo?**
  - Contando os pulsos que aparecem em um sinal de relógio de entrada que possui um período/frequência conhecido.
  - **Exemplo:** Se a frequência do relógio é igual a 1 MHz (período 1 μs) e são contabilizados 2000 pulsos, o tempo decorrido é igual a 2 ms.
- **Exemplo:**

  - **Faixa:** máximo intervalo de tempo que o temporizador consegue medir.
    Exemplo: $65535 \cdot 1x10^{-6} = 65{,}535$ ms

  - **Resolução:** mínimo intervalo de tempo que o temporizador consegue medir (período do relógio).
    Exemplo: 1 μs

  - O sinal top indica que a contagem máxima foi alcançada

Temporizador Básico



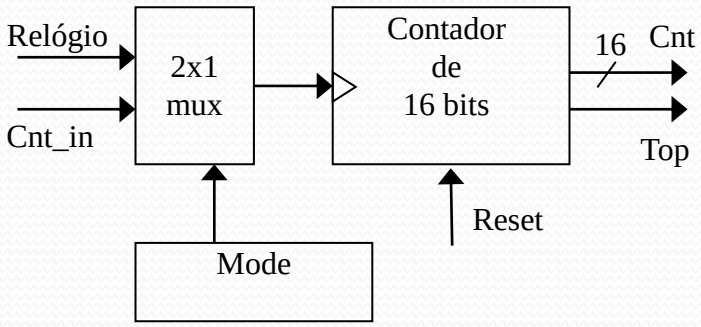
Temporizadores / Contadores

- **Contador:** mesma estrutura interna de um temporizador, porém conta pulsos gerados por outro sinal de entrada no lugar do relógio.

  - Por exemplo, a contagem de carros que passam por um sensor.

  - Com um seletor, controlado pelo modo de operação, é possível configurar um dispositivo como temporizador ou contador.

Temporizador/Contador



Temporizadores / Contadores

- **Estruturas alternativas**

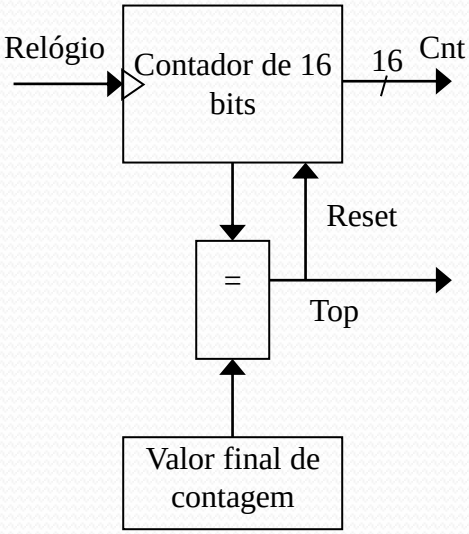
➢ O valor final da contagem é determinado de acordo com o intervalo de tempo a ser analisado. Essa estratégia permite **diminuir** a faixa do contador.

➢ Exemplo: dado o contador mostrado ao lado e um sinal de relógio com frequência de 1 MHz, qual o valor a ser armazenado como valor final de contagem para que seja medido um tempo de 4 ms?

- $4.10^{-3} / 1.10^{-6} = 4000$

➢ Por simplicidade, pode-se usar um contador descendente. Nesse caso, o comparador se resume a uma porta NOR de *n*-bits que detecta quando a contagem chega a 0.

Temporizador com um valor final de contagem



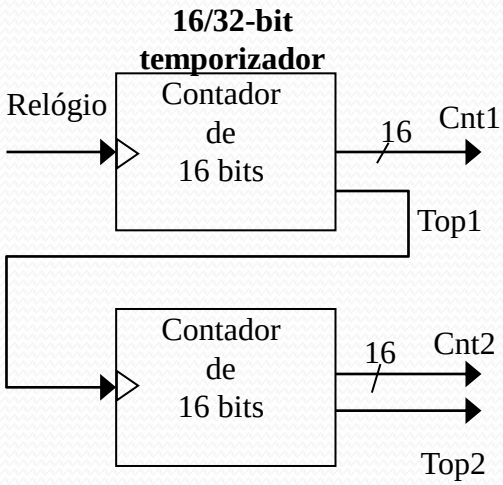
Temporizadores / Contadores

- **Estruturas alternativas**

➢Contador em série (cascata)

▪Estende o valor máximo da contagem (amplia a faixa).

▪E.g., 65536 x 65536 = 4.294.967.296.

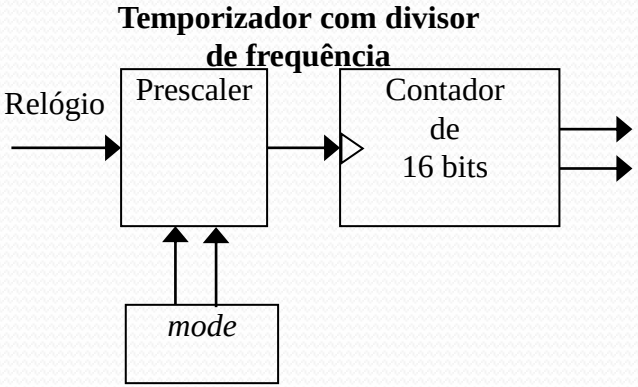


Temporizadores / Contadores

- **Estruturas alternativas**

➢Contador com divisor de frequência

▪Dependendo dos bits em *mode*, o sinal de saída do prescaler será semelhante ao relógio de entrada, mas com metade da frequência, ou um quarto da frequência, etc. Essa estratégia possibilita **aumentar** a resolução e a faixa através do aumento do período do relógio.



▪Por exemplo:

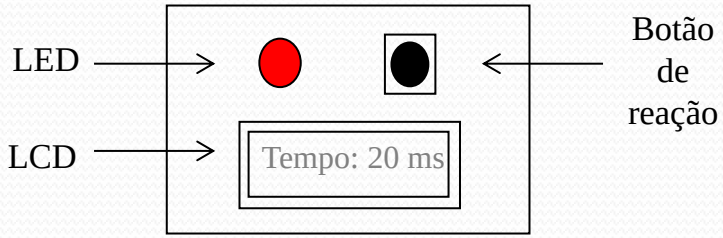
- ✓Relógio de 100 MHz.
- ✓Divisão de frequência por um fator de 8.
- ✓Relógio efetivo terá freq. de 12,5 MHz.
- ✓Resolução: 1/12,5 MHz = 80ns
- ✓Faixa: 80ns x 65535 = 5,24 ms

Temporizadores / Contadores

- É possível utilizar um processador genérico para implementar um temporizador:
  - Sabendo o número de ciclos que cada instrução consome, basta escrever um laço (loop) que executa uma determinada sequência de instruções.
  - Quando o loop se encerra, sabe-se que um determinado período de tempo (ciclos) se passou.
- Entretanto, tal estratégia pode consumir parcela significativa do tempo de processamento, diminuindo o tempo disponível para outras computações.
- Nesse sentido, é vantajoso usar um temporizador para executar a tarefa.

Temporizadores / Contadores

- Exemplo: tempo de reação



- Mede o tempo decorrido entre o acendimento do LED e o acionamento do botão por parte do usuário.
- Processador genérico de 12 MHz com temporizador de 16 bits embutido.
- A cada ciclo de instrução (igual a seis ciclos de relógio), o temporizador é incrementado.
  - A cada 6 x 1/12MHz = 0,5 μs, o valor da contagem (16 bits) é incrementado. Esta é a resolução do temporizador.
  - Faixa: 65535 x 0,5 μs = 32,77 ms
- Exige-se que o programa determine o tempo de reação com precisão de ms, e espera-se que o tempo de reação seja da ordem de segundos.

Temporizadores / Contadores

- Exemplo: tempo de reação
  - A faixa do temporizador não é suficiente para medir um tempo da ordem de segundos.
  - **Ideia:**
    - Inicializar um valor no contador de 16 bits tal que a contagem final (top = 1) seja atingida após 1 ms.
      - ✓ c = 1 ms / 0,5 μs = 2000.
      - ✓ O valor da contagem é, portanto, 65535 – 2000 = 63535.
    - Resta ao programa contar o número de vezes que o sinal top é igual a 1 para determinar o tempo de reação.

Temporizadores / Contadores

- Exemplo: tempo de reação

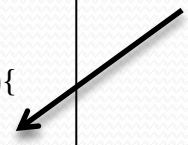
```
/* main.c */

#define MS_INIT      63535
void main(void){
  int count_milliseconds = 0;

  configure timer mode
  set Cnt to MS_INIT

  wait a random amount of time
  turn on indicator light
  start timer

  while (user has not pushed reaction button){
    if(Top) {
      stop timer
      set Cnt to MS_INIT
      start timer
      reset Top
      count_milliseconds++;
    }
  }
  turn light off
  printf("time:  %i ms", count_milliseconds);
}
```



Toda vez que top = 1, o contador (*hardware*) recomeça a contagem do valor MS_INIT e incrementamos o contador de milissegundos (*software*).

Temporizador Watchdog

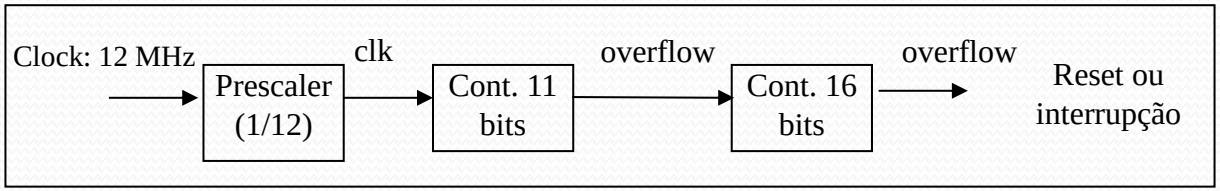
- Em vez do temporizador gerar um sinal a cada *T* unidades de tempo, é necessário que o usuário (programa) gere um sinal para o contador a cada *T* unidades de tempo.

- Caso isso não aconteça, o temporizador "expira" e gera um sinal indicando que uma falha aconteceu.

- **Usos:** detecção de falhas / auto-reset.

Temporizador Watchdog

- Exemplo: *timeout* (sinal de tempo esgotado) em um caixa eletrônico.



  - Deseja-se determinar o valor inicial a ser carregado no contador de 16 bits de tal forma que o sinal de overflow seja disparado após 2 minutos.

Temporizador Watchdog

- Exemplo: *timeout* (sinal de tempo esgotado) em um caixa eletrônico.
  - Contador de 11 bits:
    - Relógio: 12 MHz x (1/12) = 1 MHz – período = 1 μs.
    - Faixa: 2^{11} x 1 μs = 2,048 ms.
    - No 2048º ciclo, o contador de 11 bits zera e o contador de 16 bits deve incrementar seu valor.
  - Contador de 16 bits:
    - Faixa: 0 a 65535.
    - Como o valor é incrementado a cada 2,048 ms, para se atingir 2 minutos, é preciso contar $120/(2{,}048 \times 10^{-3}) = 58594$.
    - Logo, o valor inicial no contador de 16 bits deve ser:

$$65535 - 58594 = 6941.$$

Modulação por largura de pulso

- Modular: inserir informações em um sinal mediante a modificação de suas características. Exemplos: AM e FM.

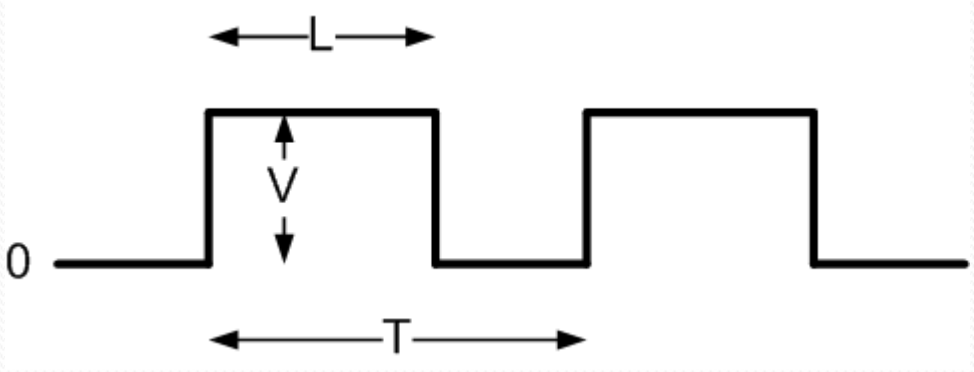
- Modulação por largura de pulso ou *Pulse Width Modulation* (PWM) ajusta o tempo que um sinal periódico com dois níveis (digital), alto e baixo, permanece em nível alto de acordo com a informação a ser inserida.

- Tempo de nível alto: largura do pulso.

- *Duty cycle*: razão entre a largura do pulso (tempo em nível alto) e período da onda.

Modulação por largura de pulso

- $duty\ cycle = \frac{L}{T}$



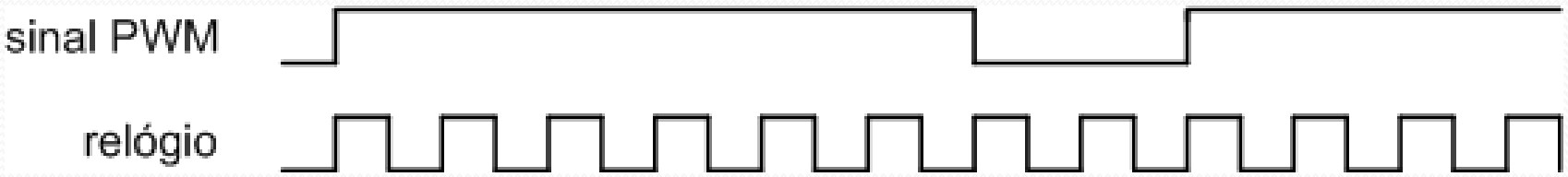
- Tensão média:

$$V_m = \frac{1}{T}\int_{t-T/2}^{t+T/2} V(t)dt = \frac{1}{T}LV = V \times duty\ cycle$$

Modulação por largura de pulso

- Implementação digital: um relógio é usado como base para estabelecer o período e a largura de pulso.

- **Exemplo:** período corresponde a 8 pulsos e tempo de nível alto a 6 pulsos de relógio.

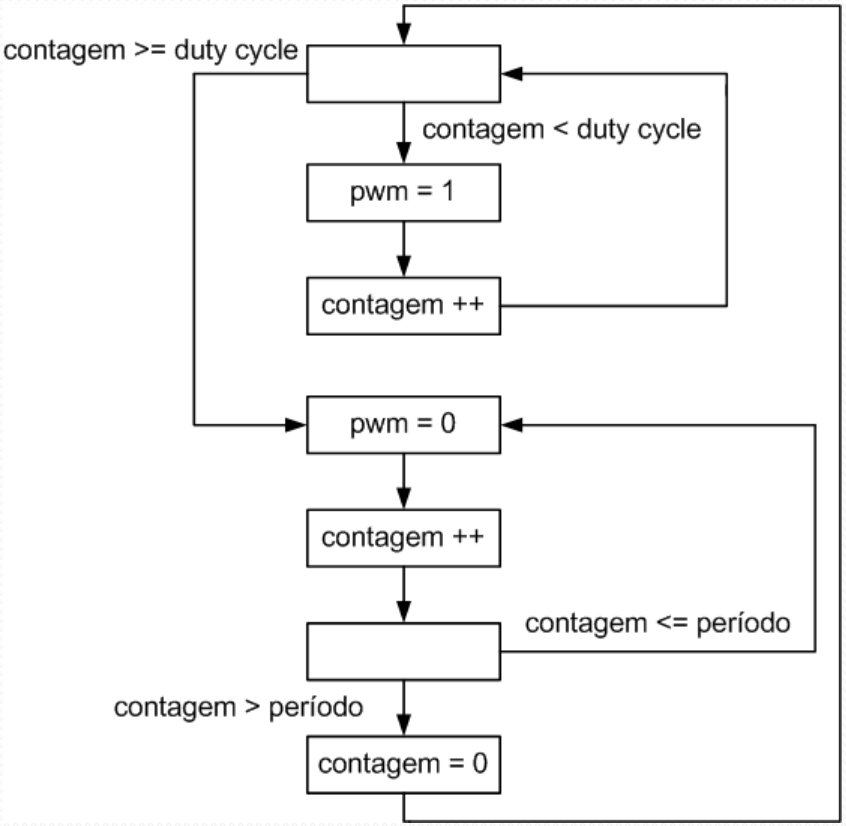


- *Duty cycle*: 6/8 = 0,75 ou 75%

- Resolução: 1/8 = 0,125

Modulador de largura de pulso

- É possível descrever a implementação digital de um PWM por meio de uma máquina de estados finitos com *datapath*:



Modulador de largura de pulso

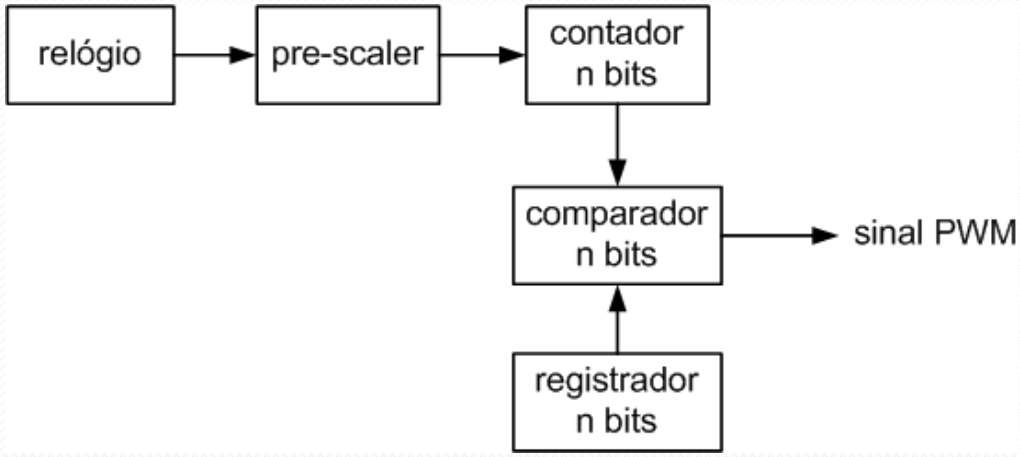
- É possível utilizar uma abordagem baseada em *software* para gerar um sinal PWM.

- **Exemplo:** Instruções ARM7TDMI

```
                MOV R5,#100     % Endereço do dispositivo de saída
                MOV R6,#1       % Nível alto
                MOV R7,#0       % Nível baixo
                MOV R9,#100     % Período
                MOV R8,#25      % Duty cycle
CICLO:          MOV R0,#0
                STR R6,[R5]     % Saída em nível alto
ALTO:           CMP R0,R8
                BEQ BAIXO
                ADD R0,R0,#1
                B ALTO
BAIXO:          STR R7,[R5]     % Saída em nível baixo
BAIXO1:         CMP R0,R9
                BEQ CICLO
                ADD R0,R0,#1
                B BAIXO1
```

Modulador de largura de pulso

- Por outro lado, uma solução baseada em *hardware* (processador dedicado) se mostra bastante natural.



- Relógio: base de tempo.
- Pre-scaler: divisor de frequência para ajuste do período do sinal PWM.
- Se a contagem é menor que o valor armazenado no registrador, o nível do sinal PWM é alto.
- Se a contagem é maior ou igual ao valor armazenado no registrador, o nível do sinal PWM é baixo.
- Portanto, o valor armazenado no registrador controla o *duty cycle*.

Modulador de largura de pulso

- O divisor de frequência pode ser programável de modo a permitir o ajuste do período do sinal PWM.
- O registrador pode ser acessado externamente e seu valor ajustado de forma a se obter o *duty cycle* desejado.
- Seja D o valor carregado no registrador. Neste caso, o *duty cycle* do sinal de saída é $\frac{D}{2^n}$.
- A resolução do *duty cycle* é dada por $\frac{1}{2^n}$.

Modulador de largura de pulso

- O período do sinal PWM é dado pelo número de estados do contador (faixa de contagem) vezes o período do relógio do contador:

$$T = 2^n \times T_{RC}$$

- A largura do pulso é dada pelo valor carregado no registrador vezes o período do relógio do contador:

$$L = D \times T_{RC}$$

Aplicações

- Controle de velocidade de motores CC, como, por exemplo, ventoinhas de computadores.
- Controle de luminosidade de lâmpadas em dimmers.
- Posicionamento de servomotores em robótica e em dispositivos radio-controlados.
- Síntese de sinais analógicos.

Controle de Motor CC

- A velocidade de um motor de corrente contínua (CC) é proporcional à sua tensão de alimentação.

- É possível controlar a velocidade, alimentando-se o motor com um sinal PWM – necessidade de driver de corrente.

- Neste caso, o *duty cycle* pode ser ajustado de modo a se obter uma tensão média correspondente à velocidade desejada.

Controle de Motor CC

- Considere um motor com carga constante que apresenta as seguintes relações entre a tensão de entrada e sua frequência de rotação:

Tensão de entrada	% da máxima tensão aplicada (duty cycle)	RPM do motor DC
0	0	0
2.5	50	4600
3.75	75	6900
5.0	100	9200

- Modulador com n = 8 bits e tensão em nível alto V=5V.

- Para que o motor gire a 4600 RPM:

$$\frac{D}{2^8} * 100 = 50 \rightarrow D = 128$$

- Para que o motor gire a 6900 RPM:

$$\frac{D}{2^8} * 100 = 75 \rightarrow D = 191$$

Dimmer

- Outra aplicação clássica de moduladores de largura de pulso é na implementação de dimmers, ou seja, de reguladores do nível de uma fonte luminosa artificial (uma lâmpada, um LED etc.).

- Nesse caso, consideremos que se tenha uma tensão máxima DC V_{MAX}, associada à máxima iluminação permitida para a fonte luminosa.

Dimmer

- É possível obter níveis de luminosidade menores regulando a tensão de alimentação através do *duty cycle*, de modo análogo ao que foi visto para o controle do motor DC.

- Deve-se ter especial cuidado com a frequência da onda quadrada utilizada – ela deve ser alta o suficiente para que se tenha uma sensação visual de continuidade.

Posicionamento de Servomotores

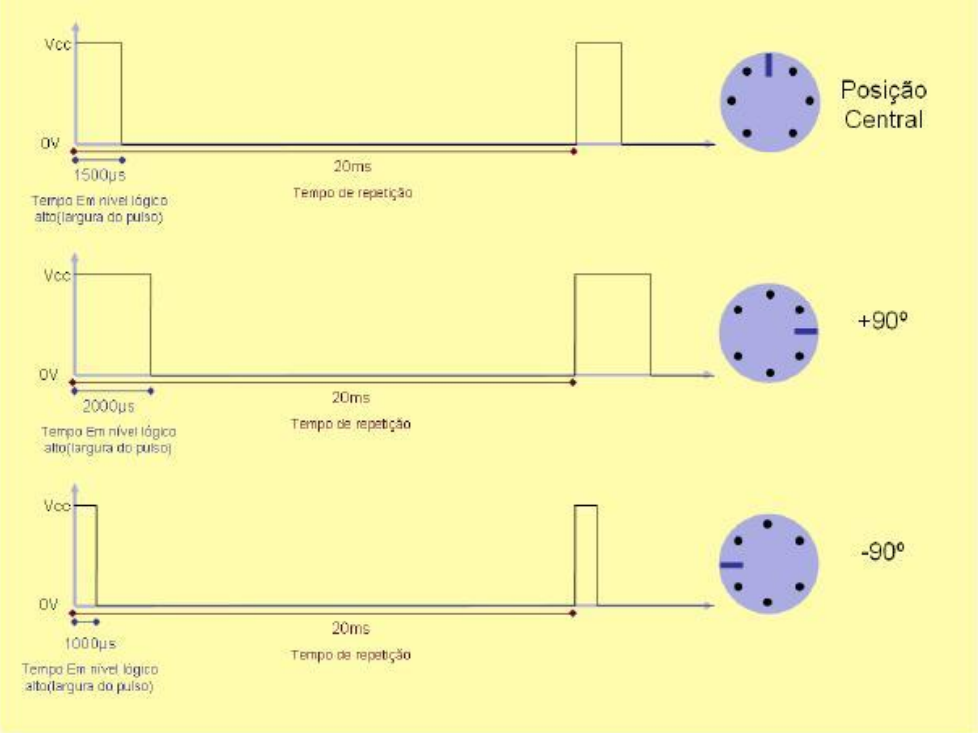
- Servomotores são atuadores rotativos que permitem um controle preciso de sua posição.

- A posição do eixo é definida pela largura do pulso de controle que deve ser enviado a uma taxa pré-definida, ou seja, um sinal PWM.

- O *pre-scaler* deve ser ajustado de acordo com a taxa desejada e o valor carregado no registrador deve atender à largura de pulso correspondente à posição.

Posicionamento de Servomotores

- **Exemplo:**
  - Modulador com n = 16bits
  - Frequência do relógio: 13,1 MHz
  - Especificações do servomotor:



Posicionamento de Servomotores

- **Primeiro passo:** determinar o fator de escala a ser usado no *pre-scaler*.
  - O ciclo de contagem deve durar 20ms.

$$T = (2^n) \times T_{RC} = 20 \rightarrow T_{RC} = \frac{20}{2^{16}} = 0{,}3051\mu s$$

  - Isto equivale a uma frequência de 3,28MHz.
  - Dividindo a frequência do relógio por esse valor:

$$\frac{13{,}1}{3{,}28} \approx 4$$

  - Ou seja, o pre-scaler precisa ser configurado de modo a dividir a frequência do relógio por 4.

Posicionamento de Servomotores

- **Segundo passo:** determinar o valor a ser carregado no registrador para se obter a posição desejada:
  - Posição central → L = 1,5 ms:

$$duty\ cycle = \frac{L}{T} = \frac{D}{2^n} \rightarrow \frac{1{,}5}{20} = \frac{D}{2^{16}}$$

$$D \approx 4915 = 1333_h$$

  - +90º → L = 2 ms:

$$\frac{2}{20} = \frac{D}{2^{16}} \rightarrow D \approx 6554 = 199Ah$$

  - -90º → L = 1 ms:

$$\frac{1}{20} = \frac{D}{2^{16}} \rightarrow D \approx 3277 = 0CCDh$$

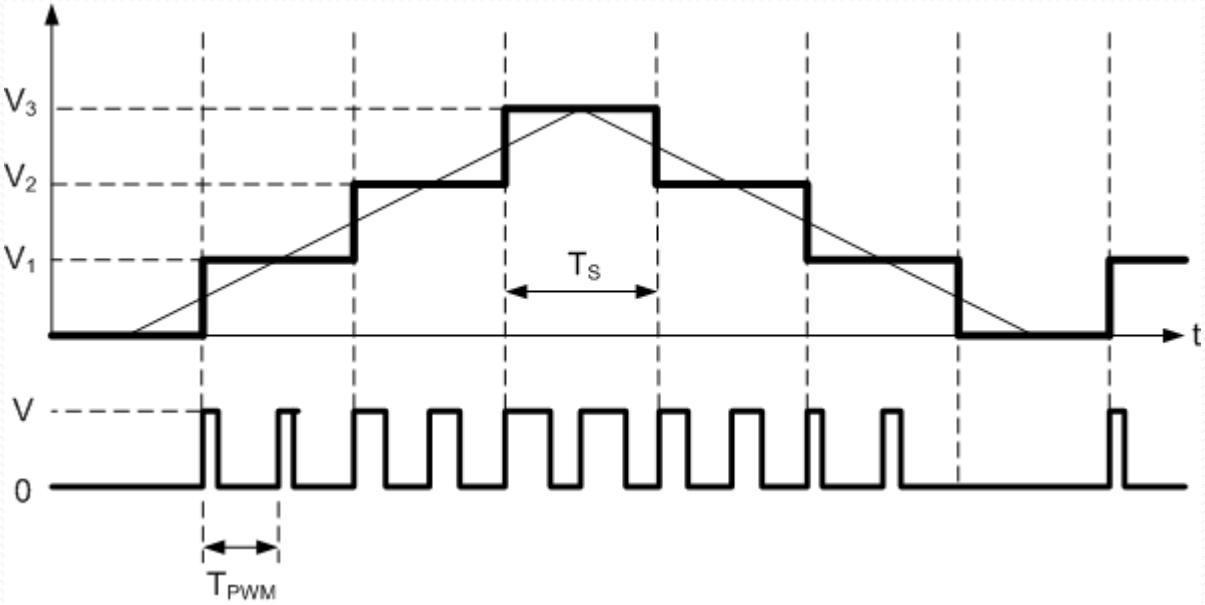
PWM - Conversão Digital-Analógica

- É possível sintetizar um sinal analógico arbitrário a partir de um sinal PWM adequadamente filtrado.

- Princípio:
  - Gerar amostras discretas do sinal desejado a intervalos de tempo regulares (taxa de amostragem).
  - Em cada intervalo de tempo, a amplitude das amostras é definida a partir da tensão média do sinal PWM, ou seja, a partir do *duty cycle*.
  - Aplicar um filtro passa-baixas para remover as altas frequências do sinal PWM (demodulação).

PWM - Conversão Digital-Analógica

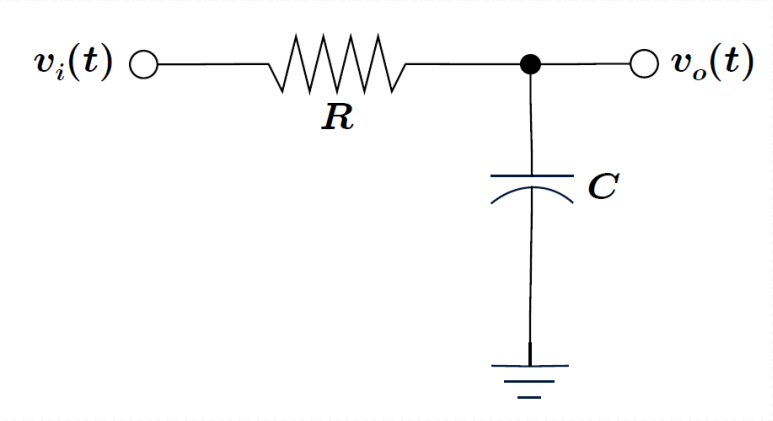
- **Exemplo:** sinal periódico triangular



- Intervalo de amostragem: T_S (Taxa de amostragem: $f_S=1/T_S$)
- f_S tem que ser maior ou igual ao dobro da frequencia máxima do sinal a ser amostrado (teorema da amostragem).
- Período do sinal PWM: T_{PWM} ($f_{PWM} = 1/T_{PWM}$)
- Amplitude da amostra: $V \times duty\ cycle$

PWM - Conversão Digital-Analógica

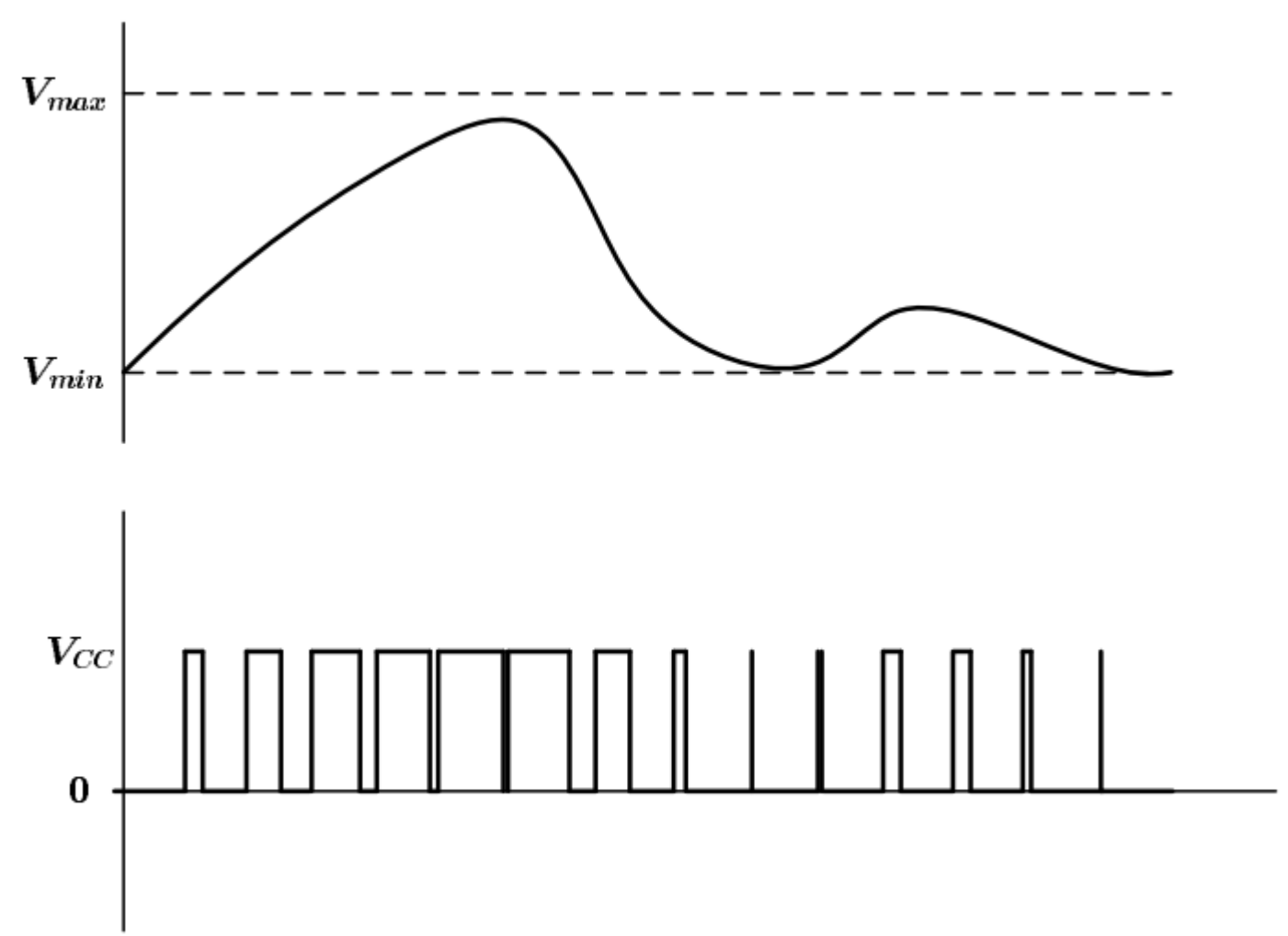
- Demodulação: conversão do sinal PWM no sinal analógico desejado.
- Filtro passa-baixa RC:



- Frequência de corte: $f_c = 1/(2\pi RC)$
- A frequência máxima do sinal analógico (f_A) deve ser muito menor que a frequência de corte do filtro.
- A frequência do sinal PWM deve ser muito maior que a frequência de corte do filtro.

$$f_A \ll fc \ll fPWM$$

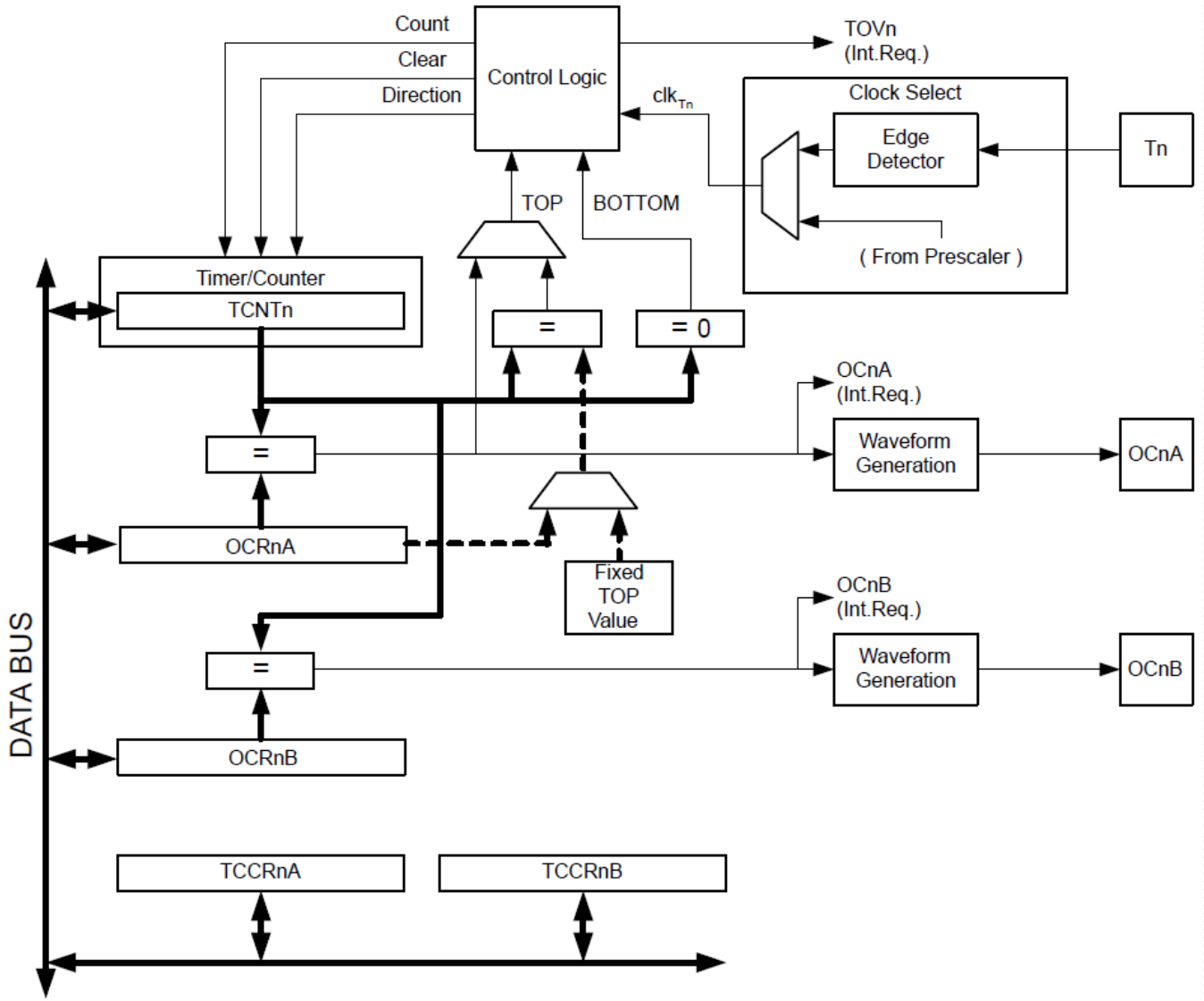
PWM - Conversão Digital-Analógica



ATmega328P - Temporizadores

- Possui três temporizadores, dois de 8 bits e um de 16 bits.
- Cada temporizador pode funcionar também como modulador PWM ou contador.
- Vários modos de operação configuráveis através de registradores de controle.
- Pre-scaler de 10 bits: pode-se usar a própria frequência do relógio do processador, 16MHz (sem pre-scaler), ou dividi-la por 8, 64, 256 ou 1024.

ATmega328P - Temporizadores



$n \in \{0,1,2\}$

ATmega328P - Temporizadores

- **Modos de operação:**
  - **Normal:** é o modo de temporização mais simples. O contador realiza uma contagem crescente de 0 a $2^n - 1$, em que n é o número de bits do contador. Ao atingir o valor máximo, a contagem é reiniciada e um *flag* é ativado sinalizando o *overflow* do contador.
  - **Clear Timer on Compare Match (CTC):** nesse modo, o registrador OCRnA (*Output Compare Register*) é usado para definir a faixa de contagem do contador. Quando a contagem alcança o valor armazenado em OCRnA, o contador é zerado e um *flag* é ativado.

ATmega328P - Temporizadores

- **Modos de operação:**
  - **Fast PWM:** é a opção para geração de sinais PWM que permite atingir as maiores frequências. Nesse modo, a contagem pode ser configurada para ir de 0 a TOP, em que TOP pode ser escolhido ou como o valor máximo de contagem, $2^n - 1$, ou definido pelo valor armazenado no registrador OCRnA. É possível gerar até dois sinais PWM com *duty cycles* distintos e mesma frequência. O duty cycle é controlado através dos registradores OCRnA e OCRnB.

ATmega328P - Temporizadores

- **Modos de operação:**
  - **Phase Correct PWM:** é a opção que possibilita a geração de sinais PWM com frequências mais baixas. Nesse modo, o contador primeiro realiza uma contagem crescente até TOP e em seguida inicia uma contagem decrescente até 0, gerando a sequência 0,1,...,TOP-1,TOP,TOP-1,...,1,0,.... O valor TOP pode ser configurado ou como $2^n - 1$ ou definido pelo valor armazenado no registrador OCRnA. Como no modo Fast PWM, também é possível gerar até dois sinais PWM com *duty cycles* distintos e mesma frequência. O duty cycle é controlado através dos registradores OCRnA e OCRnB.

ATmega328P - Temporizadores

- **Exemplo: modos de operação do temporizador 2:**

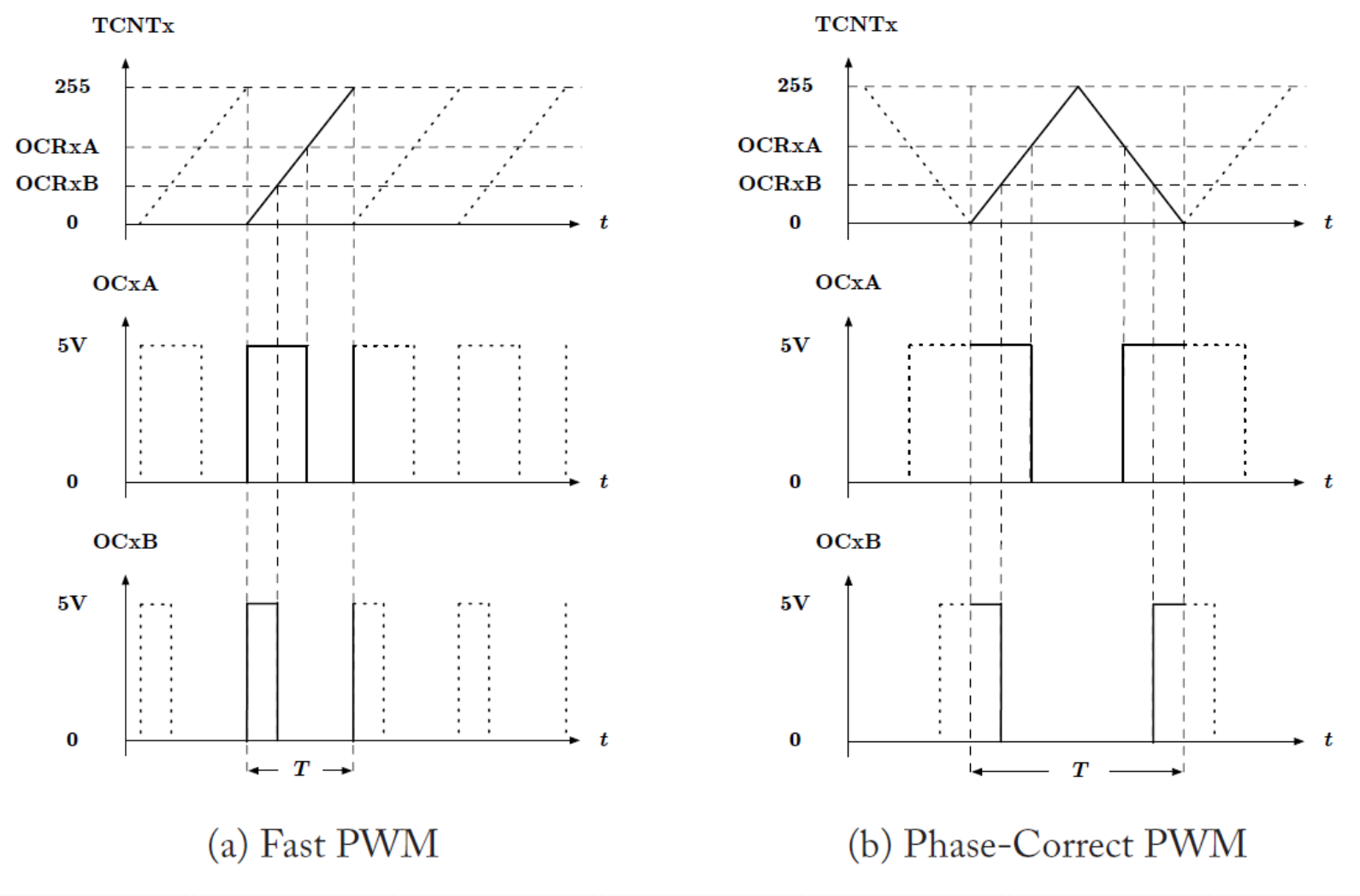
Mode	WGM22	WGM21	WGM20	Timer/Counter Mode of Operation	TOP	Update of OCR0x at	TOV Flag Set on(1)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Note:

1. MAX = 0xFF
2. BOTTOM = 0x00

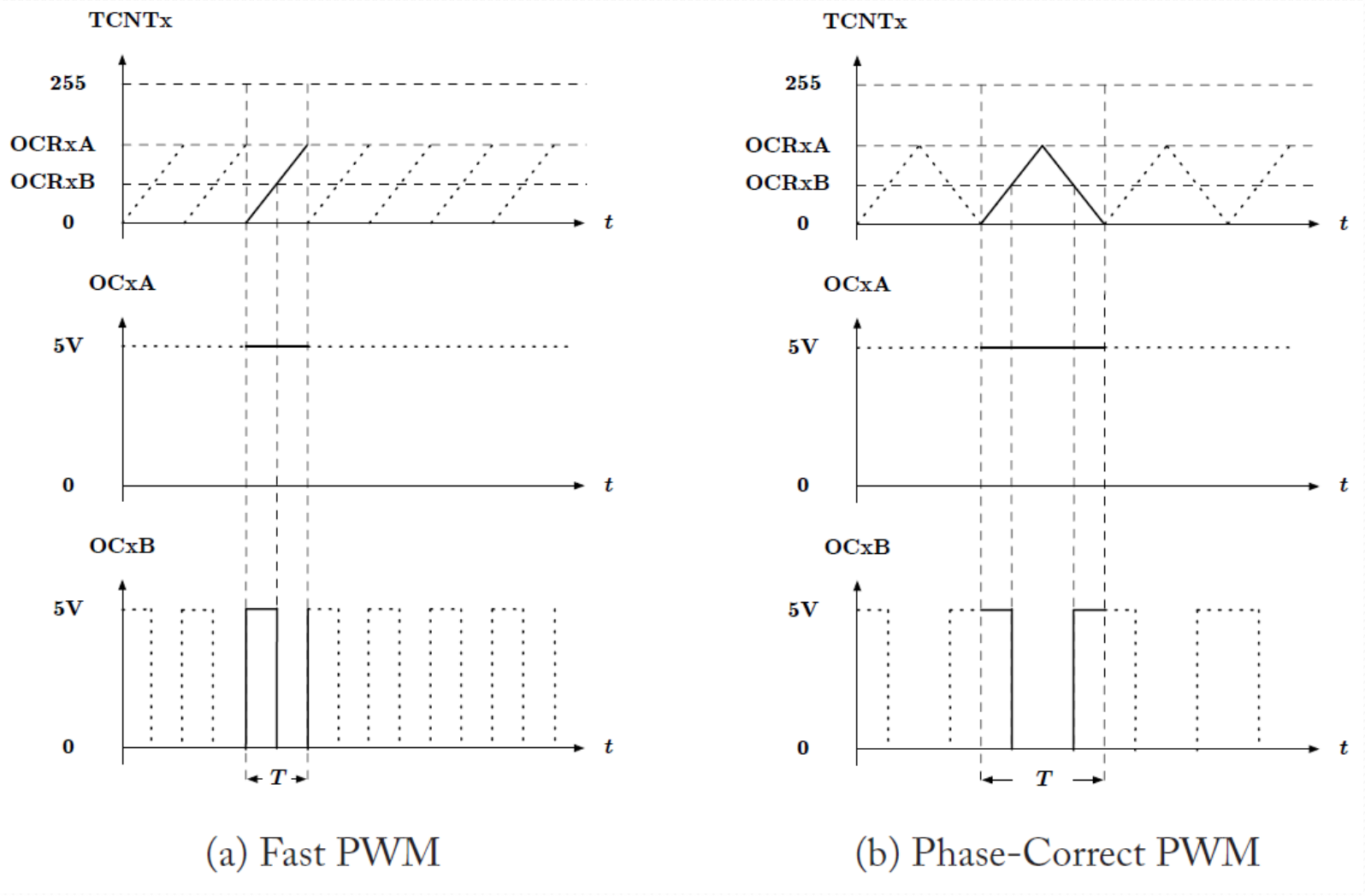
Faixa de contagem

ATmega328P - PWM



X = 0 (Timer 0), 1 (Timer 1) ou 2 (Timer 2)

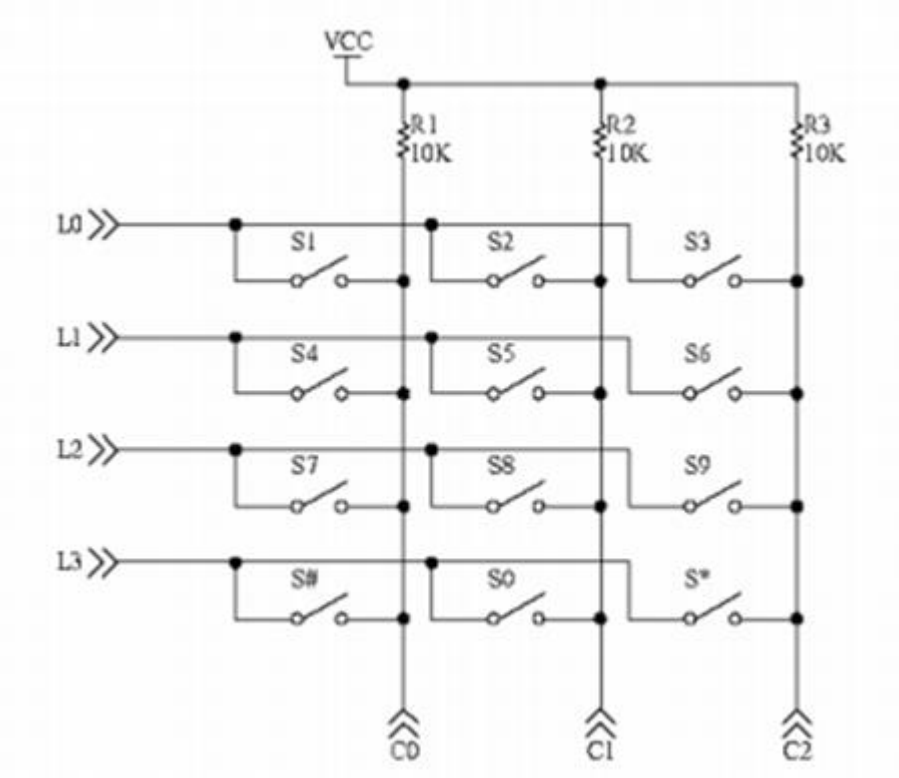
ATmega328P - PWM



X = 0 (Timer 0), 1 (Timer 1) ou 2 (Timer 2)

Controlador de teclado

- **Teclado – Matriz de chaves**



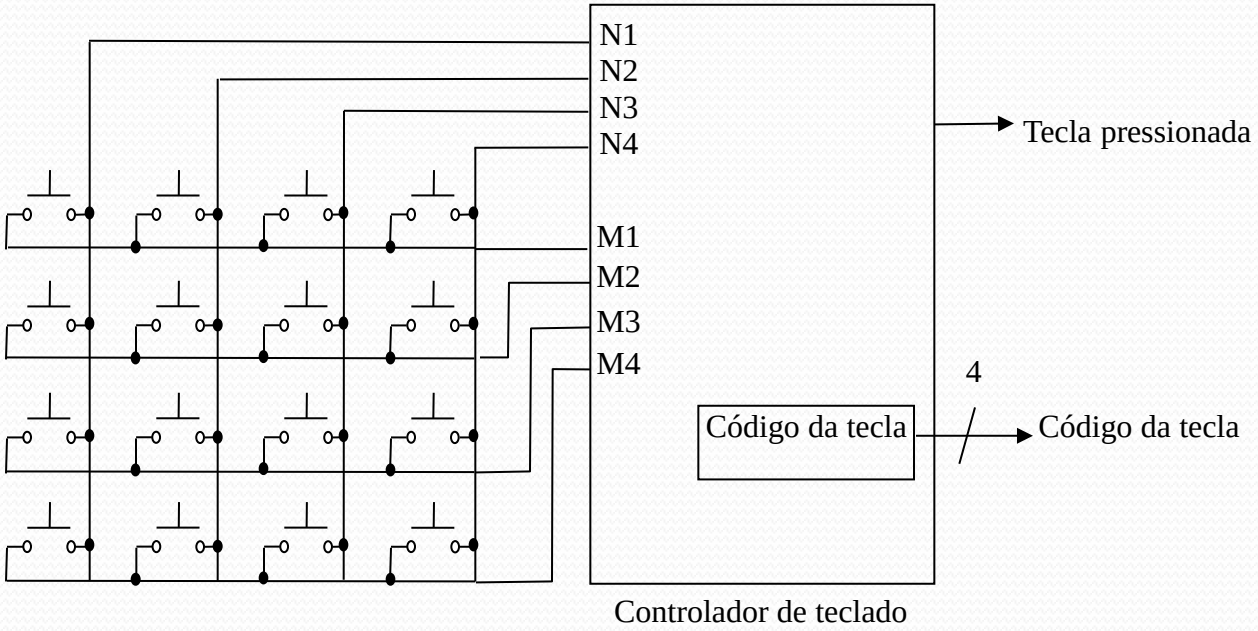
➢ As colunas (C0-C2) ficam em nível lógico alto enquanto nenhuma tecla é acionada.

➢ Para identificar uma tecla acionada, faz-se uma **varredura** da matriz do teclado.

1. Coloca-se em nível baixo apenas a primeira linha enquanto as demais ficam em nível alto.
2. Verifica-se o nível de cada coluna. Se alguma coluna estiver em nível baixo significa que a chave que conecta a coluna e a primeira linha foi pressionada.
3. Repete-se o processo para cada uma das linhas.

Controlador de teclado

- Oferece uma interface simples para leitura de caracteres de entrada em um teclado.
- O sinal "tecla pressionada" pode disparar uma interrupção, de modo que o processador genérico conectado ao controlador realiza a leitura do código da tecla.

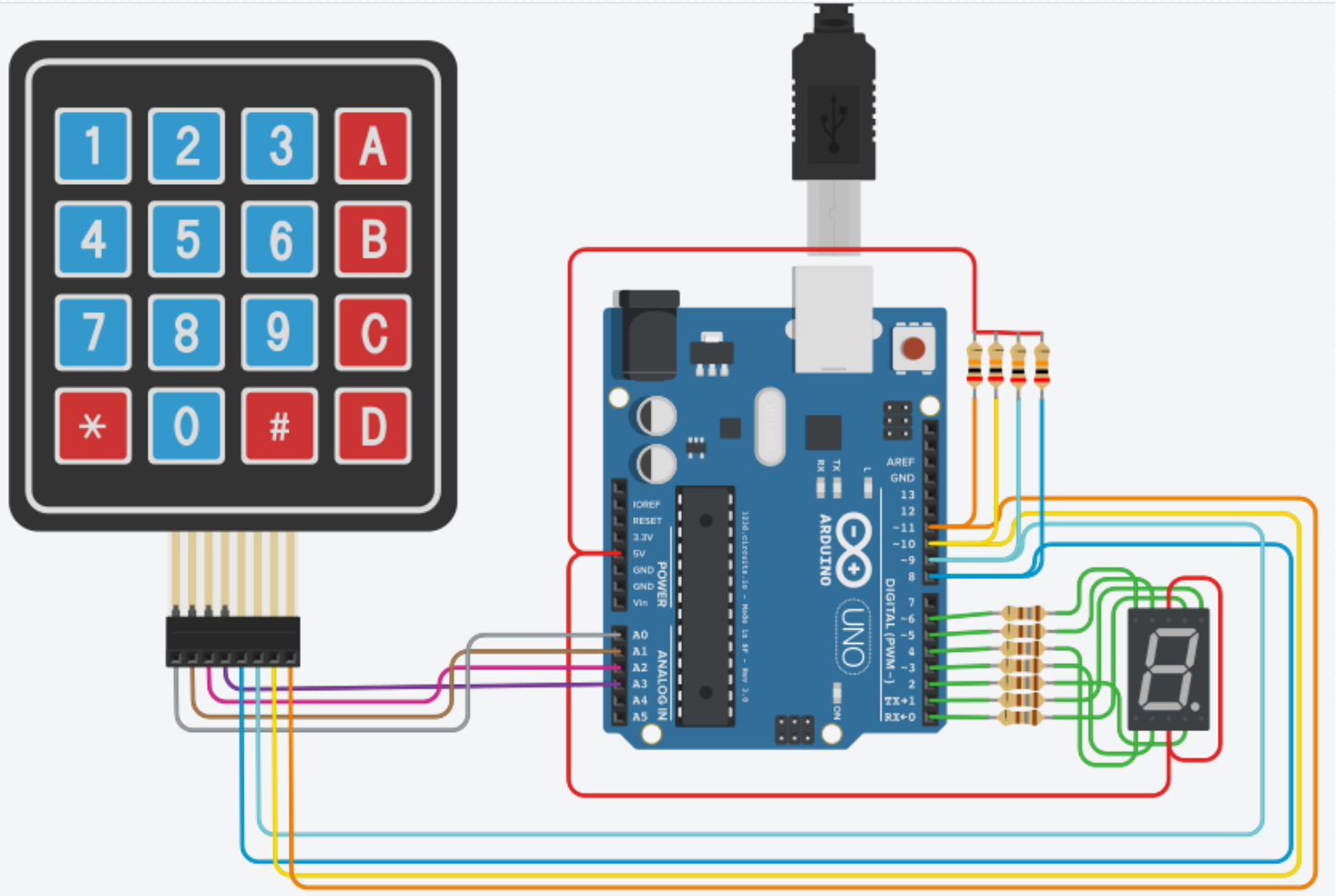


N=4, M=4

Controlador de teclado

- **Exemplo:** Varredura de teclado usando o Arduino Uno

https://circuits.io/circuits/2981152-teclado-e-display-de-7-segmentos



Controlador de teclado

- **Exemplo: Varredura de teclado usando o Arduino Uno**

  - **Linhas do teclado:** bits 0 a 3 da porta C (saídas)
  - **Colunas do teclado:** bits 0 a 3 da porta B (entradas)
  - **Display de 7 segmentos:** bits 0 a 6 da porta D estão conectados aos segmentos A a G do display (saídas)

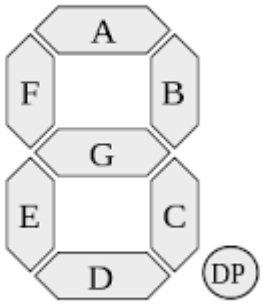
Controlador de teclado

- **Exemplo: Varredura de teclado usando o Arduino Uno**

```
#define SEG0 0b00111111
#define SEG1 0b00000110
#define SEG2 0b01011011
#define SEG3 0b01001111
#define SEG4 0b01100110
#define SEG5 0b01101101
#define SEG6 0b01111101
#define SEG7 0b00000111
#define SEG8 0b01111111
#define SEG9 0b01101111
#define SEGA 0b01110111
#define SEGB 0b01111100
#define SEGC 0b00111001
#define SEGD 0b01011110
#define SEGE 0b01111001
#define SEGF 0b01110001

unsigned char tecla;
unsigned char display[16] = {SEG1, SEG2, SEG3, SEGA, SEG4, SEG5, SEG6, SEGB, SEG7, SEG8, SEG9, SEGC, SEGE, SEG0, SEGF, SEGD};
unsigned char temp;

void setup() {
  // Os bits 0-3 da porta B são configurados como entradas e estão conectados às colunas do teclado
  DDRB = 0x00;
  // Os bits 0-3 da porta C são configurados como saídas e são responsáveis pelo acionamento das linhas do teclado
  DDRC = 0x0F;
  // A porta D é configurada como saída e é usada para acionamento do display de 7 segmentos
  DDRD = 0xFF;
  // Apaga o display (Note que o display é anodo comum. Isso significa que um segmento ascende quando a porta está em nível baixo).
  PORTD = 0xFF;
}
```



Controlador de teclado

- **Exemplo: Varredura de teclado usando o Arduino Uno**

```
// the loop routine runs over and over again forever:
void loop() {
  int col;
  int row;
  for (row=0; row < 4; row++) {
    // Coloca a linha "row" em nível baixo. As demais linhas permanecem em nível alto.
    PORTC = (PORTC & 0xF0) | ~(1<<row);
    // Armazena o conteúdo da porta B para verificar se alguma tecla foi apertada
    temp = PINB;
    // Verifica se alguma tecla foi apertada: um dos bits 0 a 3 da porta D deve ser
    // zero caso alguma tecla tenha sido acionada
    if ((temp & 0x0F) != 0x0F) {
      for (col=0; col < 4; col++) {
        // Identifica qual coluna está em nível baixo.
        if (((temp>>col) & 0x01) == 0x00) break;
      }
      // Mapeia os valores da linha e coluna para uma tecla entre 0 e 15.
      tecla = row*4 + col;
      // Mostra a tecla pressionada no display.
      // (O not inverte os bits pois a codificação original havia sido definida para
      // displays catodo comum e não anodo comum)
      PORTD = ~display[tecla];
    }
```

Controlador LCD

- ***Liquid Crystal Display*:**

  - Um mostrador de cristal líquido é um dispositivo de baixo custo e baixa potência capaz de exibir texto e imagens.

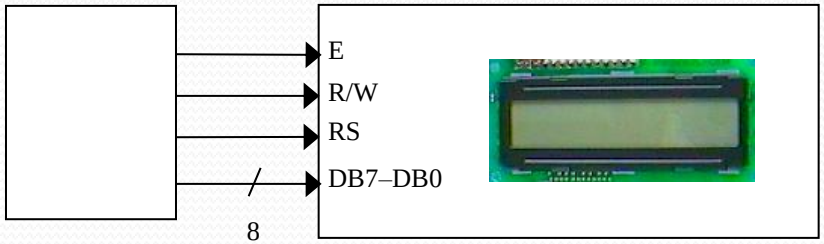
  - LCD refletivo:
    - A luz incidente passa através de uma placa de polarização. Então, a luz polarizada encontra um material cristal líquido que, quando excitado, tem suas moléculas alinhadas, o que faz com que a luz polarizada o atravesse. Senão, a luz não passa.
    - Por fim, a luz que passou atinge uma superfície refletora, de modo que estas regiões excitadas acendem.

  - Controlador LCD: provê uma interface simples para o uso de LCDs.

Controlador LCD

- **Exemplo:** controlador LCD

  - Recebe palavras de controle do microcontrolador e realiza as ações correspondentes no LCD.



  - RS: quando em nível BAIXO, informa ao controlador que os dados enviados (DB7 – DB0) formam uma palavra de controle. Quando em nível ALTO, informa ao controlador que os dados enviados correspondem ao código de um caractere que deve ser exibido no LCD.

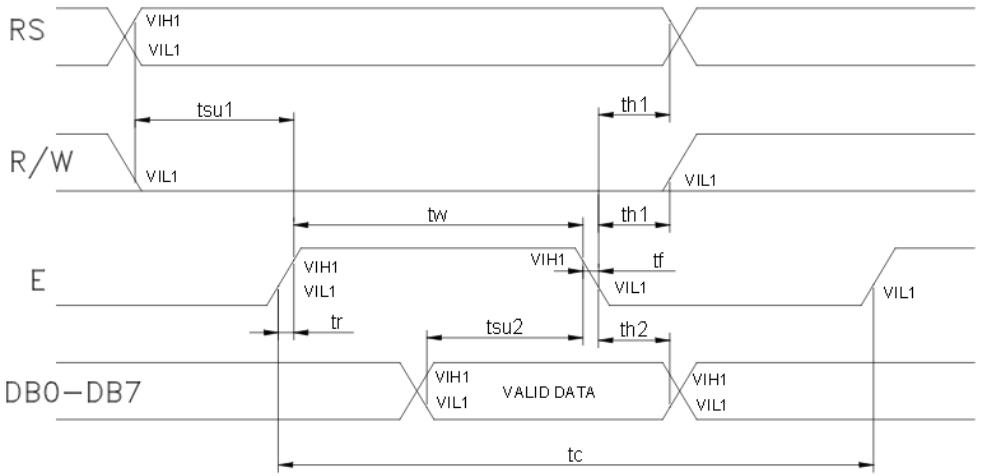
  - Toda vez que um dado deve ser enviado, o bit de *enable* (E) precisa ser acionado.

Controlador LCD

RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0	Description
0	0	0	0	0	0	0	0	0	1	Clears all display, return cursor home
0	0	0	0	0	0	0	0	1	*	Returns cursor home
0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and/or specifies not to shift display
0	0	0	0	0	0	1	D	C	B	ON/OFF of all display(D), cursor ON/OFF (C), and blink position (B)
0	0	0	0	0	1	S/C	R/L	*	*	Move cursor and shifts display
0	0	0	0	1	DL	N	F	*	*	Sets interface data length, number of display lines, and character font
1	0	WRITE DATA								Writes Data

CODES	
I/D = 1 cursor moves left	DL = 1 8-bit
I/D = 0 cursor moves right	DL = 0 4-bit
S = 1 with display shift	N = 1 2 rows
S/C =1 display shift	N = 0 1 row
S/C = 0 cursor movement	F = 1 5x10 dots
R/L = 1 shift to right	F = 0 5x7 dots
R/L = 0 shift to left	

Ciclo de escrita



Controlador LCD

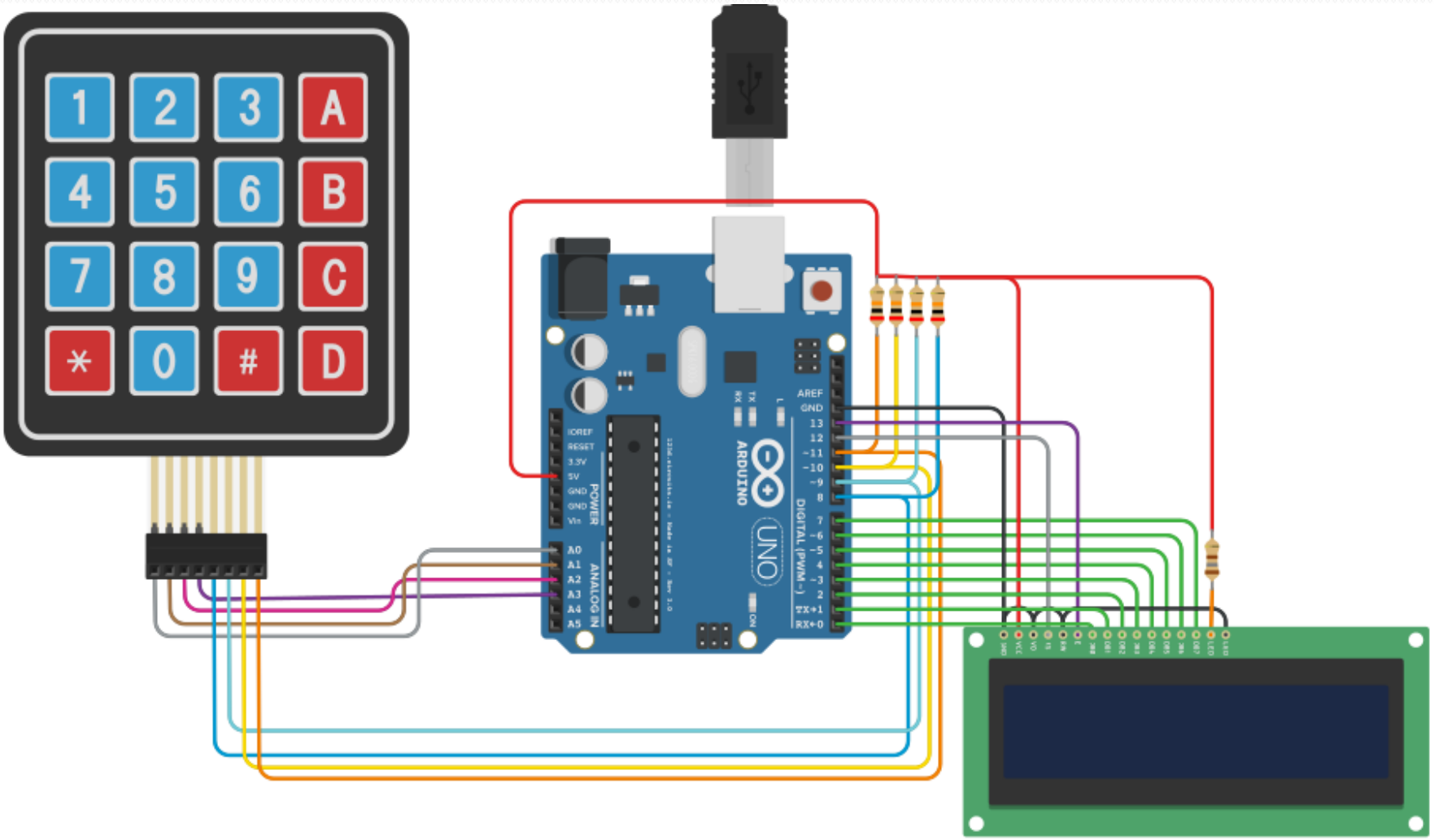
- **Mapa de caracteres:**

Há uma coleção de caracteres armazenados na memória do controlador que podem ser exibidos no LCD mediante o envio do código correspondente.

Custom Pattern	Decimal	Hex
Row 1:	4	0x04
Row 2:	14	0x0E
Row 3:	14	0x0E
Row 4:	14	0x0E
Row 5:	31	0x1F
Row 6:	0	0x00
Row 7:	4	0x04

Controlador LCD

- **Exemplo:** controlador LCD, teclado e Arduino Uno https://circuits.io/circuits/2991921-teclado-e-display-lcd



Controlador LCD

- **Exemplo:** controlador LCD, teclado e Arduino Uno
  - **Linhas do teclado:** bits 0 a 3 da porta C (saídas)
  - **Colunas do teclado:** bits 0 a 3 da porta B (entradas)
  - **Sinais de controle do LCD:** RS e E são os bits 4 e 5 da porta B (saídas)
  - **Dados do LCD:** sinais D0 a D7 são os bits 0 a 7 da porta D (saídas)

```
#define SEG0 0x30
#define SEG1 0x31
#define SEG2 0x32
#define SEG3 0x33
#define SEG4 0x34
#define SEG5 0x35
#define SEG6 0x36
#define SEG7 0x37
#define SEG8 0x38
#define SEG9 0x39
#define SEGA 0x41
#define SEGB 0x42
#define SEGC 0x43
#define SEGD 0x44
#define SEGE 0x2A
#define SEGF 0x23

unsigned char tecla;
unsigned char display[16] = {SEG1, SEG2, SEG3, SEGA, SEG4, SEG5, SEG6, SEGB, SEG7, SEG8, SEG9, SEGC, SEGE, SEG0, SEGF, SEGD};
unsigned char temp;
```

Controlador LCD

- **Exemplo:** controlador LCD, teclado e Arduino Uno

```
void setup() {
  // Os bits 0-3 da porta B são configurados como entradas
  // e estão conectados às colunas do teclado.
  // Os bits 4 e 5 da porta B são configurados como saidas
  // e são usados para controle do LCD (bit 4 - RS; bit 5 - E).
  DDRB = 0x30;
  // RS = 0; E = 0;
  PORTB = PORTB & 0b11001111;
  // Os bits 0-3 da porta C são configurados como saídas e
  // são responsáveis pelo acionamento das linhas do teclado.
  DDRC = 0x0F;
  // A porta D é configurada como saída e é usada para
  // acionamento do LCD - barramento de dados D0-D7.
  DDRD = 0xFF;
  // Inicializa o display
  initLCD();
}
```

Controlador LCD

- **Exemplo:** controlador LCD, teclado e Arduino Uno

```
// the loop routine runs over and over again forever:
void loop() {
  int col;
  int row;
  while (1) {
    for (row=0; row < 4; row++) {
      // Coloca a linha "row" em nível baixo. As demais linhas permanecem em nível alto.
      PORTC = (PORTC & 0xF0) | ~(1<<row);
      // Armazena o conteúdo da porta B para verificar se alguma tecla foi apertada
      temp = PINB;
      // Verifica se alguma tecla foi apertada: um dos bits 0 a 3 da porta D deve ser
      // zero caso alguma tecla tenha sido pressionada
      if ((temp & 0x0F) != 0x0F) {
        for (col=0; col < 4; col++) {
          // Identifica qual coluna está em nível baixo.
          if (((temp>>col) & 0x01) == 0x00) break;
        }
        // Codifica a tecla pressionada com base na coluna e linha correspondentes.
        // O código é um valor entre 0 e 15: tecla 1 -> 0, tecla 2 -> 1,..., tecla D -> 15
        tecla = row*4 + col;
        // Mostra a tecla pressionada no display.
        writeChar(display[tecla]);
        // Aguarda a tecla ser desativada
        while (((PINB>>col) & 0x01) == 0x00);
      }
    }
  }
```

Controlador LCD

- **Exemplo:** controlador LCD, teclado e Arduino Uno

```
void initLCD(void) {
  // Apaga o display e retorna o cursor para posição inicial
  // RS = 0
  PORTB = PORTB & 0b11101111;
  // Coloca o byte contendo o código do comando no barramento
  // de dados do display
  PORTD = 0x01;
  // Habilita o display (E = 1)
  PORTB = PORTB | 0x20;
  delay(1); // matem o sinal E em nível alto por 1ms
  // Desabilita o display (E = 0)
  PORTB = PORTB & 0b11011111;
  // Aguarda o término da execução da instrução
  delay(5);

  // Ativa cursor (underline indica a posição do cursor)
  PORTD = 0x0E;
  // Habilita o display (E = 1)
  PORTB = PORTB | 0x20;
  delay(1); // matem o sinal E em nível alto por 1ms
  // Desabilita o display (E = 0)
  PORTB = PORTB & 0b11011111;
  // Aguarda o término da execução da instrução
  delay(5);
}
```

```
void writeChar(unsigned char caractere) {
  // RS = 1
  PORTB = PORTB | 0x10;
  // Coloca o caractere digitado no barramento
  // de dados do display
  PORTD = caractere;
  // Habilita o display (E = 1)
  PORTB = PORTB | 0x20;
  delay(1); // matem o sinal E em nível alto por 1ms
  // Desabilita o display (E = 0)
  PORTB = PORTB & 0b11011111;
  // RS = 0
  PORTB = PORTB & 0b11101111;
  // Aguarda o término da execução da instrução
  delay(5);
}
```

UART

- **UART (Universal Asyncronous Receiver/Transmitter):** receptor/transmissor universal assíncrono.

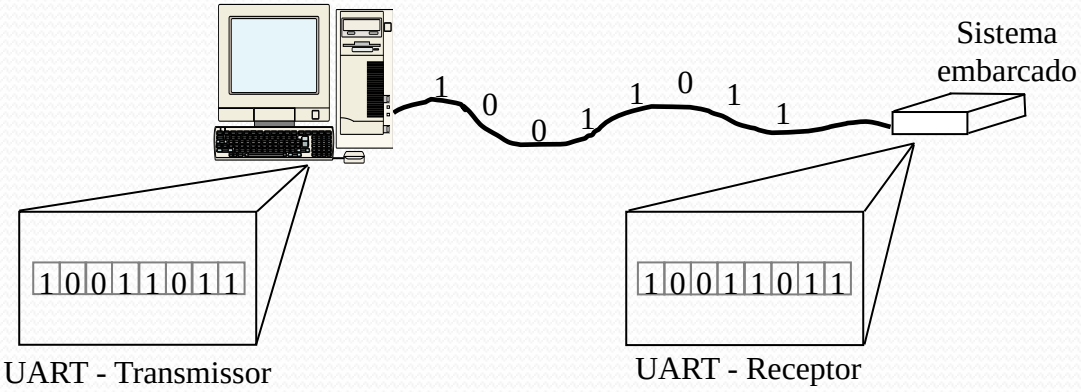
- Recebe dados seriais e os armazena como dados paralelos (usualmente um *byte*); também recebe dados paralelos e os transmite de forma serial.

- Útil para comunicação entre dispositivos bastante afastados ou quando há poucos pinos de entrada/saída disponíveis.

- *Baud rate* = determina a velocidade com que dados são trocados entre duas UARTs; mede o número de mudanças de sinal por segundo que ocorrem na linha serial.

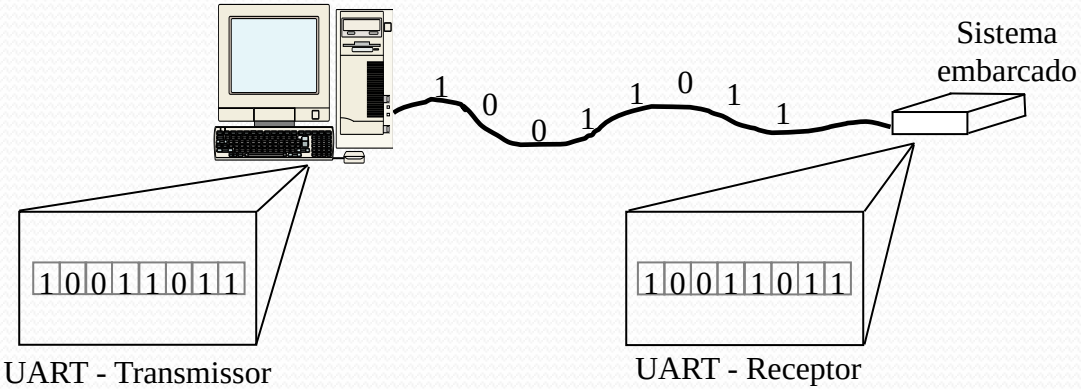
- **Protocolo de comunicação:** receptor e transmissor concordam com
  i. taxa em que bits são enviados (*baud rate*);
  ii. número de bits usados para separar duas transmissões consecutivas (*stop bits*);
  iii. número de bits de dados;
  iv. (opcional) tipo de paridade (usado para verificação de erros).

UART



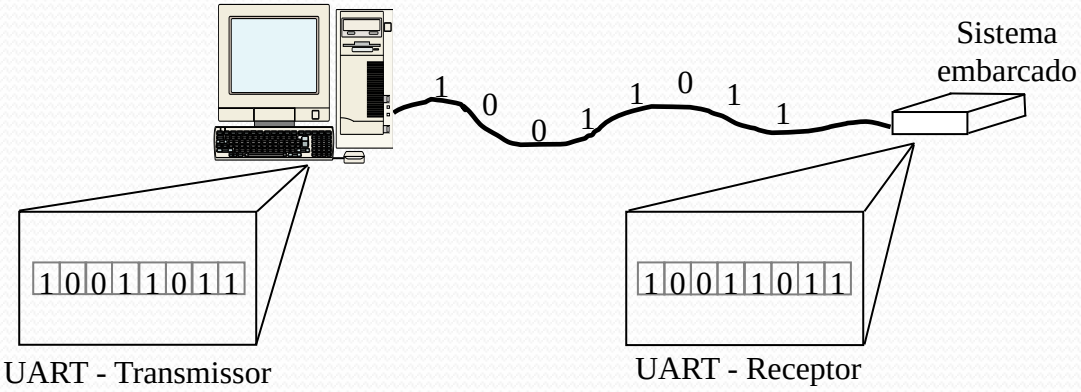
- **Recepção:**
  - O receptor está constantemente monitorando o pino de entrada à espera de um bit de partida (e.g., uma transição de ALTO para BAIXO). Após esse bit, o receptor amostra o pino em intervalos de tempo pré-determinados, deslocando cada bit amostrado em um registrador de deslocamento de recepção.

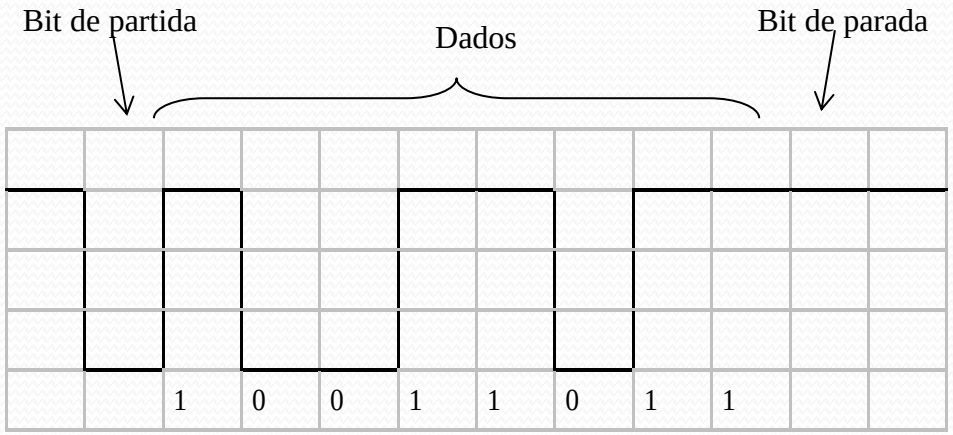
UART



- **Transmissão:**
  - O processador escreve um *byte* no registrador da UART, a qual inicia a transmissão com um bit de partida no pino de saída.
  - Então, o transmissor desloca os dados do registrador de deslocamento para o pino de saída de acordo com uma taxa pré-determinada. Além disso, também envia um bit de paridade (opcional) e um bit de parada, encerrando a transmissão.

UART


Sistema
embarcado
1 0 0 1 1 0 1 1
1 0 0 1 1 0 1 1
UART - Transmissor
UART - Receptor


Bit de partida
Dados
Bit de parada
1 0 0 1 1 0 1 1

UART: Bit de Paridade

- Pode ser usado na detecção de erros de transmissão.
- Paridade ímpar (*odd*): se o número de bits "1" no dado transmitido for **par**, o bit de paridade é "1". Caso contrário, o bit de paridade é "0".

$$P_{odd} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

- Paridade par (*even*): se o número de bits "1" no dado transmitido for **ímpar**, o bit de paridade é "1". Caso contrário, o bit de paridade é "0".

$$P_{even} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$

UART – ATmega328P

- O microcontrolador ATmega328P possui uma UART. De fato se trata de uma USART (Universal Sincronous Asyncronous Receiver/Transmitter) pois ela pode ser usada também para comunicação serial síncrona (I^2C ou SPI).

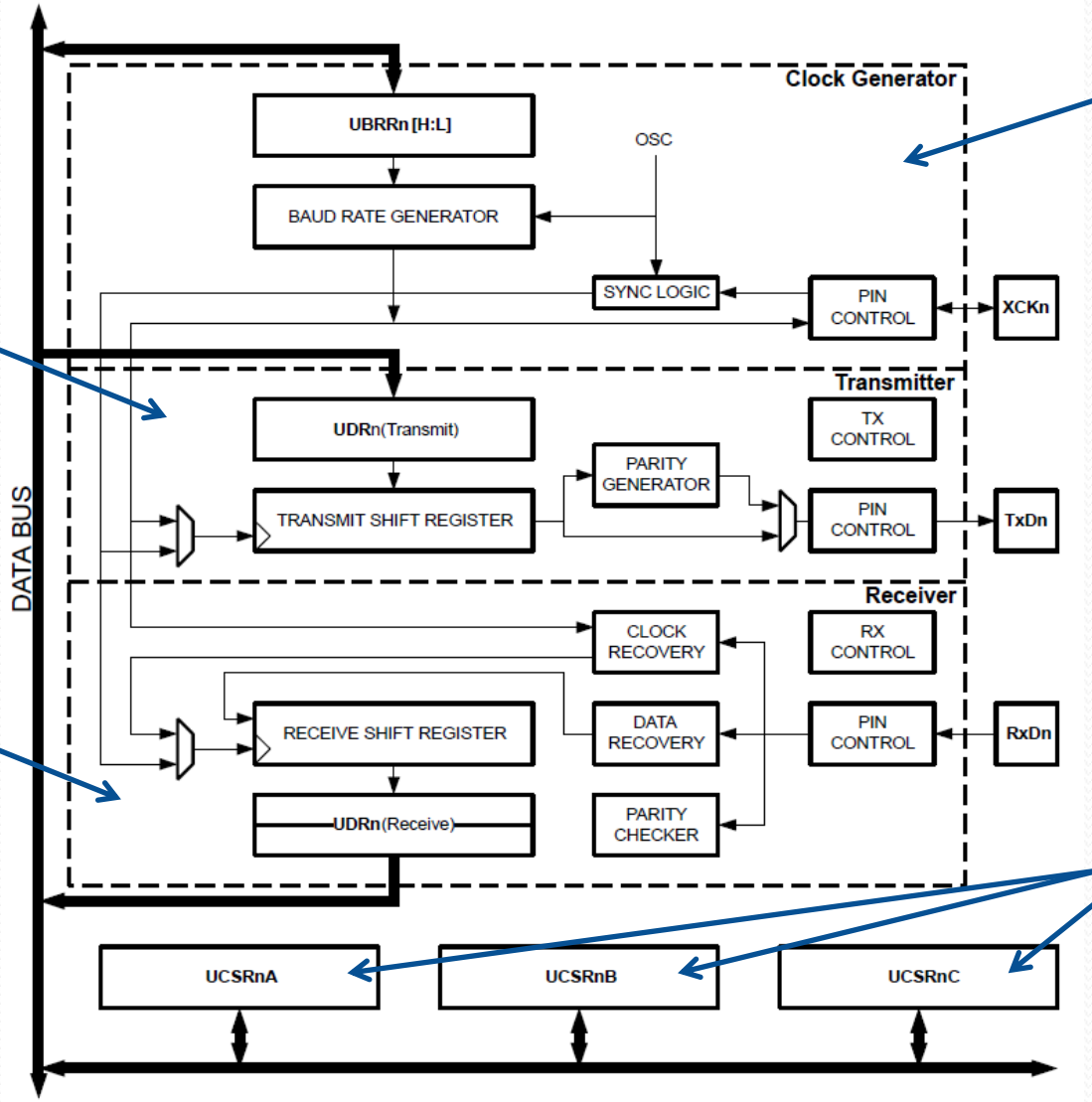
- O diagrama em blocos da USART é mostrado na figura a seguir.

UART – ATmega328P

Geração de clock (*baud rate*)

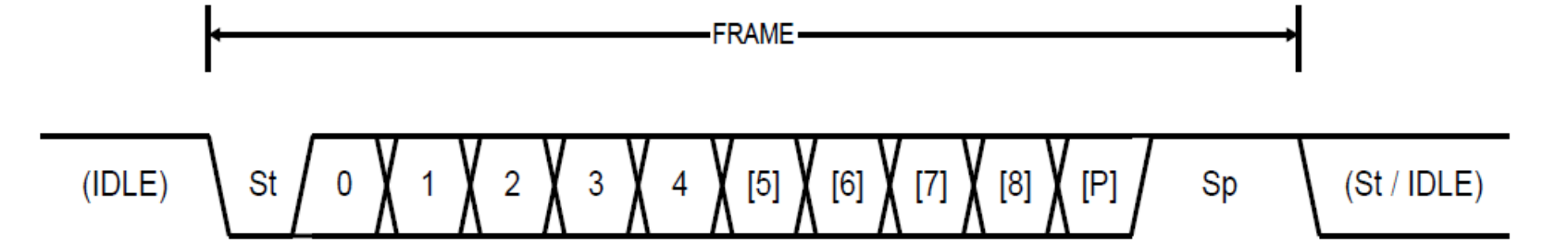
Transmissor

Receptor



Registradores de controle e status

UART – ATmega328P



- Cada bloco de dados é enviado em um pacote denominado *frame*. Os *frames* são criados a partir da concatenação de bits de sincronização e paridade aos bits de dados.
- Estrutura do *frame*:
  - **St:** O primeiro bit é o *start bit*. Ele sempre tem nível baixo.
  - **(n):** Bits de dados (0 a 8). O primeiro bit de dado é o LSB.
  - **P:** Bit de paridade (opcional). Pode ser *odd* ou *even*.
  - **Sp:** O último bit do frame é o *stop bit*. Pode haver um ou dois *stop bits* e eles sempre apresentam nível alto.

UART – ATmega328P

- **Registradores:**
  - **USART Baud Rate 0 Register Low (UBRR0L) e USART Baud Rate 0 Register High (UBRR0H):** Definem o baud rate da comunicação.

Baud Rate [bps]	f_{osc} = 16.0000MHz				f_{osc} = 18.4320MHz				f_{osc} = 20.0000MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250k	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	–	–	4	-7.8%	–	–	4	0.0%
1M	0	0.0%	1	0.0%	–	–	–	–	–	–	–	–
Max.(1)	1Mbps		2Mbps		1.152Mbps		2.304Mbps		1.25Mbps		2.5Mbps	

UART – ATmega328P

- **Registradores:**
  - **USART Control and Status Register 0 A (UCSR0A):** Contém os flags que sinalizam o recebimento ou o término da transmissão de uma palavra de dados. Também possui flags que apontam erro no frame recebido, erro de paridade e *overrun* (chegada de dados antes dos precedentes terem sido lidos). Um bit é usado para controle do *baud rate.*

  - **USART Control and Status Register 0 B (UCSR0B):** Habilita ou desabilita o transmissor e o receptor. Configura as interrupções associadas à USART. Ajusta o número de bits de dados transmitido ou recebido em cada *frame* (5 a 9) (configuração complementar ao registrador UCSR0C).

UART – ATmega328P

- **Registradores:**
  - **USART Control and Status Register 0 C (UCSR0C):** Seleciona o modo de operação da USART (síncrono ou assíncrono), o modo de paridade (desabilitado, *even* ou *odd*), o número de *stop* bits do *frame* (1 ou 2) e o número de bits de dados transmitido ou recebido em cada *frame* (5 a 9) (configuração complementar ao registrador UCSR0B).

  - **USART I/O Data Register 0 (UDR0):** Registrador de função dupla que armazena os dados recebidos e os dados a serem transmitidos. A leitura do UDR0 acessa o conteúdo do buffer de recepção. Já a escrita no UDR0 altera o conteúdo do buffer de transmissão. A escrita no buffer de transmissão só é efetuada se ele se encontrar vazio, condição sinalizada pelo flag UDRE0 do registrador UCSR0A em nível alto. Caso contrário, o dado é ignorado e não é transmitido.

UART – ATmega328P

- **Exemplo:** Inicialização

```
void USART_init(void) {
  // Configura o baud rate -> BAUD_PRESCALER=0x0067 (9600 baud)
  UBRR0 = BAUD_PRESCALER;
  UCSR0A &= ~(1<<U2X0);
  // Configura o frame: 8 bits de dados, 1 stop bit
  UCSR0C = ((0<<USBS0)|(1<<UCSZ01)|(1<<UCSZ00));
  // Habilita o receptor e o transmissor
  UCSR0B = ((1<<RXEN0)|(1<<TXEN0));
}
```

UART – ATmega328P

- **Exemplo:** Transmissão

```
void USART_send( unsigned char data) {
  // Aguarda até que o buffer de transmissão esteja vazio (UDRE0 = 1)
  while(!(UCSR0A & (1<<UDRE0)));
  // Escreve o dado a ser transmitido
  UDR0 = data;
}
```

UART – ATmega328P

- **Exemplo:** Recepção

```
unsigned char USART_receive(void) {
  // Aguarda um dado ser recebido (RXC0 = 1)
  while (!(UCSR0A & (1<<RXC0)));
  // Lê o dado recebido
  return UDR0;
}
```

UART – ATmega328P

- **Exemplo:** Comunicação serial entre duas placas Arduino Uno

https://circuits.io/circuits/3018169-comunicacao-via-uart